A blue sign with white text

Description automatically generated with medium confidence

**UNIVERSITY OF BOLTON**

**BSc COMPUTING / BEng Software Engineering**

**COURSEWORK SUBMISSION FORM**

|  |
| --- |
| **Student/Centre to complete:**    SURNAME/FAMILY NAME: SABIR FORENAMES: NAVEED    BOLTON STUDENT ID: 2224755 EMAIL: ns7crt@bolton.ac.uk    DATE OF SUBMISSION: 12/05/2023 @ 4:00 PM    MODULE NO./TITLE: SWE4201 Introduction to Software Development    TUTOR’S NAME: Abdul Razak    COURSEWORK TITLE: The Friston College  Please state if this is your FIRST submission OR REFERRED/DEFERRED submission OR a REPEAT submission?  FIRST  …………………..…………………………………………………………………………………………  **Declaration**  **I hereby declare that this work is my own work. I understand that if I am suspected of plagiarism or another form of cheating, my work be referred to Academic Registrar and/or the Board of**  **Examiners, which may result in me being expelled from the programme. I understand once I submit this work, it will automatically belong to the University of Bolton.** |

Academic staff to complete:

Feedback: …………………………………………………………………………………………………

……………………………………………………………………………………………........................

………………………………………………………………………………………………………………

Date Issued: 20 March 2023 Hand-In Date: **12 May 2023 @ 4 pm**

Other Relevant Date e.g., Demonstration: **on or before week commencing** **8 May 2023 and week**

**commencing 15 May 2023 during the practical sessions.**

Received: On Time □ Late □ (within 5 days of published deadline date)

Mark awarded: % Do not apply mark penalty unless the work was submitted late.

Assessors Name: A. Razak Signature:

Date:

Degree Conversions A: 70-100% B: 60-69% C: 50-59% D: 40-49% F: 0-39% HND Conversions Pass: 40-49% Merit: 50-66% Distinction: 67-100%

**Assignment 1 – Marking Scheme and Student Feedback**

|  |  |
| --- | --- |
| **Student ID number & Name** | 2224755  NAVEED SABIR |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FUNCTIONALITY PROVIDED** |  |  |  |  | **Max Mark** |
| User Requirements Set A | 1 | a | b | c | 39% |
| 2 | a | b | c |
| User Requirements Set B | 3 | a | b | c | 49% |
| 4 | a | b | c |
| User Requirements Set C | 5 | a | b | c | 59% |
| User Requirements Set D | 6 | a |  |  | 89% |

***Circle the letters showing what functionality you have implemented.***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PROGRAMMING CODE** | **BS** | **WA** | **A** | **G** |
| Appropriate user defined classes provided |  |  |  |  |
| Classes have appropriate fields |  |  |  |  |
| Classes have appropriate constructors |  |  |  |  |
| Classes have appropriate accessor & mutator methods |  |  |  |  |
| Classes have other methods as required |  |  |  |  |
| Java naming convention used for classes, fields and methods |  |  |  |  |
| Java naming convention used for parameters and variables |  |  |  |  |
| Use of descriptive variable names |  |  |  |  |
| Appropriate validation and error messages |  |  |  |  |
| Overall structure and logic of code |  |  |  |  |
| Overall quality of code |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **JAVADOC DOCUMENTATION** | **BS** | **WA** | **A** | **G** |
| All classes documented |  |  |  |  |
| All public fields and methods documented |  |  |  |  |
| All parameters and return values documented |  |  |  |  |
| Documentation clearly states the purpose of the class/method |  |  |  |  |
| Documentation clearly explains how to use the methods |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CLASS DIAGRAM** | **BS** | **WA** | **A** | **G** |
| All classes included |  |  |  |  |
| All public fields and methods included |  |  |  |  |
| All private fields and methods included |  |  |  |  |
| Correct relationships between classes |  |  |  |  |

**BS – Below acceptable standard : WA – Weak but acceptable : A – Acceptable : G – Good**

[Figure 1 7](#_Toc134765321)

Contents

[INTRODUCTION / OVERVIEW 4](#_Toc134765371)

[OBJECT-ORIENT PROGRAM 5](#_Toc134765372)

[CHALLENGES FACED 6](#_Toc134765373)

[UML CLASS DIAGRAM 8](#_Toc134765374)

[ADDITIONAL FUNCTIONALITIES 9](#_Toc134765375)

[REFERENCES 10](#_Toc134765376)

# INTRODUCTION / OVERVIEW

Scenario

The village of Friston has had its own community college for 30 years, but it is now under threat of closure due to spending cuts. The only hope is to improve the college’s efficiency, and to do this they have asked you to create a computer program to keep track of its students and courses.

Overview

The purpose of this project is to develop a software system that represents things such as Student, Courses, Enrolment, and College using object-oriented programming approaches. The school keeps basic information about each student and the offered courses, such as the course title, registration cost, and unique identity number assigned to each course and student. The institution records the student ID number, course ID number, and date of enrolment when a student enrols in a course. Four components must be tried to complete the software development process: programme implementation, class diagram design, Javadoc documentation, and testing. The program's implementation aims to create a software system that meets the capabilities defined in the college system user criteria. The Class diagram describes the implementation, and the Javadoc documentation will properly explain the programme implementation. Finally, testing plans to develop new classes for evaluating college functionality as well as methods for testing implementation.

# OBJECT-ORIENT PROGRAM

Object-oriented programming concepts are a collection of ideas that drive the design and implementation of object-oriented software systems. This assignment makes use of the following object-oriented programming concepts:

1. Encapsulation: The implementation of classes and their methods is intended to hide the system's implementation details from the outside world. Objects' internal states are concealed, and access to them is controlled through methods (Sahni, 2014) Selic, B. (2008) .
2. Inheritance: Through inheritance, classes may be created that inherit features and functions from parent classes. This aids in the creation of reusable and maintainable code (Nahin, 2018).
3. Polymorphism: Objects' capacity to take on several forms enables flexibility and extensibility in software system architecture
4. Abstraction: The abstraction notion enables the development of abstract classes and interfaces that specify object behaviour and characteristics. This helps to isolate the system's technical details from its operation (Unknown, 2021).
5. Modularity: By employing modular design principles, the system is divided into smaller, self-contained components. This makes it easier to maintain and test the system Smith, J. (2018) Chen, J., & Singh, R. (2016).

Overall, the use of these object-oriented programming principles helps in creating a software system that is flexible, maintainable, and extensible.

# CHALLENGES FACED

GENERAL DIFFICULTIES

1. Time management: Programming tasks can be time-consuming and demand a great deal of concentration and attention to detail. Balancing professional demands with other academic obligations and commitments may be tough.
2. Debugging: Finding and correcting errors in your code may be time-consuming and unpleasant. It may be particularly tough if you are unfamiliar with the programming language or environment in which you are working.
3. Programme design: It may be difficult to create a programme that achieves the objectives of the assignment while also being efficient, scalable, and maintainable.

SPECIFIC COMPLICATIONS

* 1. Programming execution:

Potential roadblocks to programme implementation include:

* Inadequate understanding of user requirements
* Difficulty in converting user specifications into actual code.
* Bugs or flaws in the code that affect how the software functions.
* Inefficient code that slows down programme execution
* Challenges faced to integrate increasingly complex user-supplied functionality.
* Issues finishing the *student enrolment by date range* requirement.
* Issues with the main menu and the functionalities inside, specifically the if statements
* Issues with the methods in *college* class such as building a proper functionality for the requirements with a combined use of getter and setter methods from different classes.
* Issued faced with *student ID* and *course ID* as the methods would not operate properly because of declaration.
  1. Classification diagram:
* Here are a few potential roadblocks to building a class diagram:
* Difficulty in picking proper classes to reflect the program's functionality.
* Class connection misconceptions, such as inheritance or composition
* Incorrectly expressing functionality on the class diagram, resulting in programme implementation issues
* Because of poor class design, maintaining and upgrading the software becomes more challenging.

1. Javadoc documentation:

* When creating Javadoc documentation, the following concerns may arise:
* Uncertainty about how to utilise Eclipse's documentation tool.
* Difficulty precisely detailing the functioning and procedures of the programme.
* Inconsistent or insufficient documentation, making it difficult for others to understand or use the programme.
* Changes or enhancements to the program's functionality are not documented, which causes confusion or problems.

1. Testing: Possible difficulties for testing the program could include:
   * Difficulty in defining the testing classes required to adequately test the program's functionality.
   * Bugs or faults in the testing class(es) create incorrect or false positive findings.
   * Difficulty in developing thorough test cases that cover every imaginable scenario.
   * Inefficient testing approaches that slow down the testing process
   * Failure to understand how to use Javadoc to document the testing class(es).

BUSINESS ASPECT

Implementing a software engineering solution in a commercial setting can be complex and time-consuming. Some are:

1. Resource constraints:

It is possible that a significant quantity of resources will be required to develop a software engineering solution. Competent employees, equipment, and infrastructure are required to design, test, and deploy software. Budget and schedule restrictions, for example, might make the implementation process difficult.

1. Integration with existing systems:

Software engineering solutions must commonly integrate with existing systems in a business environment. Understanding the present system architecture, establishing acceptable connection points, and ensuring that the new solution does not disturb existing operations may all be demanding chores.

1. User adoption:

Typically, user approval is crucial to the success of a software engineering solution. If clients are reluctant to utilise the new system or find it difficult to use, the solution may fall short of expectations.

1. Maintenance and support:

After building a software engineering solution, it must be maintained and supported on an ongoing basis. This might be a big barrier for firms, especially those with little IT resources.

To overcome these obstacles, a well-defined implementation strategy that encompasses requirements gathering, resource allocation, testing, training, and continuous maintenance and support is necessary. Engaging stakeholders and users throughout the implementation phase may also aid in ensuring that the solution satisfies their requirements and is effectively implemented.

# UML CLASS DIAGRAM

A UML class diagram is a graphical representation of a software system's classes, interfaces, and interactions. It is an excellent tool for software developers to utilise while creating and debating software designs. The UML class diagram depicts the system graphically, allowing developers to identify classes and their links to other classes Rumbaugh, J., Jacobson, I., & Booch, G. (2005).

Classes, attributes, methods, and links are all represented in the class diagram. A class is a design template for constructing objects; it has attributes (data members) and methods (member functions) that define how the object behaves. The attributes represent the class's properties, whereas the methods define the actions that the class may do Oestereich, B. (2002).

The links between the classes are illustrated in the figure by arrows, which reflect the many types of ties that exist between the classes. The three basic types of links are inheritance, composition, and association. The term "inheritance" refers to a "is-a" connection in which a subclass inherits the properties of its superclass. Composition represents a "has-a" connection, which happens when one class includes one or more items from another class. The phrase "association" refers to a relationship that does not imply ownership or limitation Larman, C. (2004).

Overall, the UML class diagram provides a high-level view of a software system, allowing engineers to better comprehend and describe the structure of the system. It facilitates in the identification of essential classes and their relationships, making software system creation and management easier Muller, P. A. (2001).

A picture containing text, diagram, parallel, font

Description automatically generated

Figure

# ADDITIONAL FUNCTIONALITIES

Scanner for user input:

A scanner class is included in the functionality, allowing the programme to take human input. The user can engage with the software and supply input that the programme can utilise to conduct actions or make judgements. The scanner class accepts input from the keyboard or other sources and scans a variety of data types like as integers, floating-point numbers, and text.

Menu based on Scanner:

There is also a menu-based interface that takes use of the scanner class. This allows the user to select from a menu of options, following which the software does the appropriate action. A menu-based interface may improve user interaction usability and intuitiveness.

System.out.format for tables:

To format console output, such as tabular data displays, use the System.out.format function. This approach may be used to align and format data to make it more intelligible and orderly, especially when dealing with large amounts of data. This might be a useful addition to the assignment if the programme has to display data in a neat and ordered manner, such as when displaying a list of students or courses.

# REFERENCES

Sahni, S. (2014). Data structures, algorithms, and applications in java (2nd ed.). S.l.: Jones & Bartlett Learning.

Selic, B. (2008). Modularity in object-oriented systems. Journal of Object Technology, 7(8), 21-34. https://www.jot.fm/issues/issue\_2008\_09/article1/

Nahin, P. J. (2018). The Science of Computer Programming. Springer International Publishing.

Unknown. (2021). Principles of Object-Oriented Programming. Guru99. <https://www.guru99.com/object-oriented-programming-oops-concept.html>

Smith, J. (2018). Modularity in software design: breaking down a system into smaller, self-contained parts. TechBeacon. https://techbeacon.com/app-dev-testing/modularity-software-design-breaking-down-system-smaller-self-contained-parts

Chen, J., & Singh, R. (2016). An overview of modularity in software design. International Journal of Advanced Research in Computer Science and Software Engineering, 6(8), 73-79. <http://www.ijarcsse.com/docs/papers/Volume_6/8_August2016/V6I8-0111.pdf>

Rumbaugh, J., Jacobson, I., & Booch, G. (2005). The unified modeling language reference manual (2nd ed.). Addison-Wesley.

Oestereich, B. (2002). UML class diagrams. Object-Oriented Technology: Concepts and Principles, 239-273.

Muller, P. A. (2001). UML distilled: A brief guide to the standard object modeling language. Addison-Wesley.

Larman, C. (2004). Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development. Prentice Hall.